Mission Control GUI Guide

Table of Contents

[1 Introduction 2](#_Toc42708330)

[1.1 Qt Version Selection 2](#_Toc42708331)

[1.2 Open Source License Obligations 2](#_Toc42708332)

[1.3 Compiler Selection 2](#_Toc42708333)

[1.4 Language Selection 2](#_Toc42708334)

[1.5 Design Notes 3](#_Toc42708335)

[2 Reference Documentation 3](#_Toc42708336)

[2.1 Qt Framework Documentation 3](#_Toc42708337)

[2.2 Qt Creator IDE Manual 3](#_Toc42708338)

[2.3 Qt Installation 3](#_Toc42708339)

[2.4 Heob for Qt 3](#_Toc42708340)

[3 Development Environment Setup 4](#_Toc42708341)

[3.1 Install Qt and Qt Creator 4](#_Toc42708342)

[3.2 Install Heob 5](#_Toc42708343)

[4 Loading the Project File 6](#_Toc42708344)

[5 Building and Running the Project 6](#_Toc42708345)

[6 Analyzing with Heob 7](#_Toc42708346)

[7 Target Color 7](#_Toc42708347)

[8 Target Shape 8](#_Toc42708348)

[9 Target Dimensions 8](#_Toc42708349)

[10 Task List 8](#_Toc42708350)

[11 Mission Export File 8](#_Toc42708351)

# Introduction

This guide discusses the basic steps of modifying, compiling, and testing the initial version of the Robosub Mission Control GUI. The GUI and guide were developed on a Windows operating system, but the executable was deployed for Linux and a Raspberry Pi 4. This guide will be written from the perspective of a developer creating on a Windows machine.

The intent of this document is to provide useful information for developers. It is assumed that the reader has a general understanding of the Qt 5 widget toolkit and C++ programming language.

## Qt Version Selection

At the time of development, Qt 5.12.2 was the most recent stable version of the Qt Framework. It is important to note that widgets or tools from Qt 3 or prior are considered depreciated and not compatible with Qt 5.

## Open Source License Obligations

The Robosub Mission Control GUI was developed using an open source Qt license. To use this license, Qt requires the following obligations to be observed:

* Provide a re-linking mechanism for Qt libraries
* Provide a license copy & explicitly acknowledge Qt use
* Make a Qt source code copy available for customers
* Accept that Qt source code modifications are non-proprietary
* Make “open” consumer devices
* Accept Digital Rights Management terms, please see the GPL FAQ
* Take special consideration when attempting to enforce software patents FAQ

## Compiler Selection

The MinGW compiler was selected because it provides a complete open source programming tool set which is suitable for the development of native Microsoft Windows applications. Most importantly, MinGW does not depend on any 3rd-party C-Runtime DLLs. MinGW is also compatible with Qt Creator. See installation details for more information.

## Language Selection

The Qt 5.12.2 framework was built in C++ and is most compatible with the C and C++ 11 standard libraries. Therefore, C++ 11 coding standards should be observed during development. It is also important to note that Qt has special Qt-specific libraries that directly mirror C++ standard libraries. Qt prefers Qt, so it is recommended that instead of a std class, use the Qt equivalent. For example, instead of <string> use <QString>. See section 2.1 Qt Framework Documentation for a link to Qt 5 online documentation.

C++ provides many features that makes code faster to develop, easier to maintain, and significantly safer than C code (usually without any loss in performance). These features include data encapsulation (classes), pass by reference, scoped names, strong types, automatic type detection, strongly typed enumerations, constant correctness, etc. Using these features helps avoid many of the hard to detect programming errors typical in C applications. Therefore, the code will be implemented using C++ 11.

## Design Notes

The software architecture includes the follow features:

1. The program will be written in C++ and have an object-oriented design.
2. The application is intended to be run on a Raspberry Pi 4.

# Reference Documentation

## Qt Framework Documentation

Qt 5.14 documentation is available at <https://doc.qt.io/qt-5/index.html>.

All Qt classes can be found here: <https://doc.qt.io/qt-5/classes.html>

All Qt modules can be found here: <https://doc.qt.io/qt-5/qtmodules.html>. This application primarily employs the Qt Core, Qt GUI, and Qt Widgets modules.

QML reference documentation can be found here: <https://doc.qt.io/qt-5/qmlreference.html>

## Qt Creator IDE Manual

Qt Creator is an IDE that includes UI designer tools and on-device debugging. For more information, visit the online manual page: <https://doc.qt.io/qtcreator/index.html>

## Qt Installation

Section 3.1 details how to use the Qt Online Installer to install Qt 5.14.2. More information on this process and links to how to install Qt using binaries can be found on the Open Source Qt downloads website: <https://www.qt.io/download-open-source>

## Heob for Qt

Heob is a heap observer for detecting buffer overruns and memory leaks built for the Windows operating system. It is recommended to use Heob over Valgrind because Heob is integrated in to and can be run from Qt Creator. Additionally, to run Valgrind on Windows, you need to run it with an external application. After downloading Heob, it’s a relatively straight forward to use just by clicking Analyze from the Qt Creator menu bar.

**QT GOTCHA:** Be wary when investigating memory leaks. Qt applications are known to throw false positives when analyzed with Heob, GDB, or Valgrind. To take advantage of Qt’s garbage collector, make sure always give an object a parent. When the parent is deleted, it will clean up its children.

To install Heob, first download its zipfile from SourceForge (<https://sourceforge.net/projects/heob/>). Heob can also be downloaded from GitHub (<https://github.com/ssbssa/heob>), however, you will have to build the source files before Heob can be used. SourceForge ships the Heob .exe for easy drop into Qt.

After downloading Heob, create a file in Qt/Tools named Heob and unzip the downloaded files here. To run Heob with MinGW, you will need to also provide Dwarfstack .dlls. Download them from here (<https://github.com/ssbssa/dwarfstack/releases>, unzip the files, and place them in the Qt/Tools/Heob folder.

For information on how to use Heob from within Qt Creator, see here: <https://doc.qt.io/qtcreator/creator-heob.html>.

# Development Environment Setup

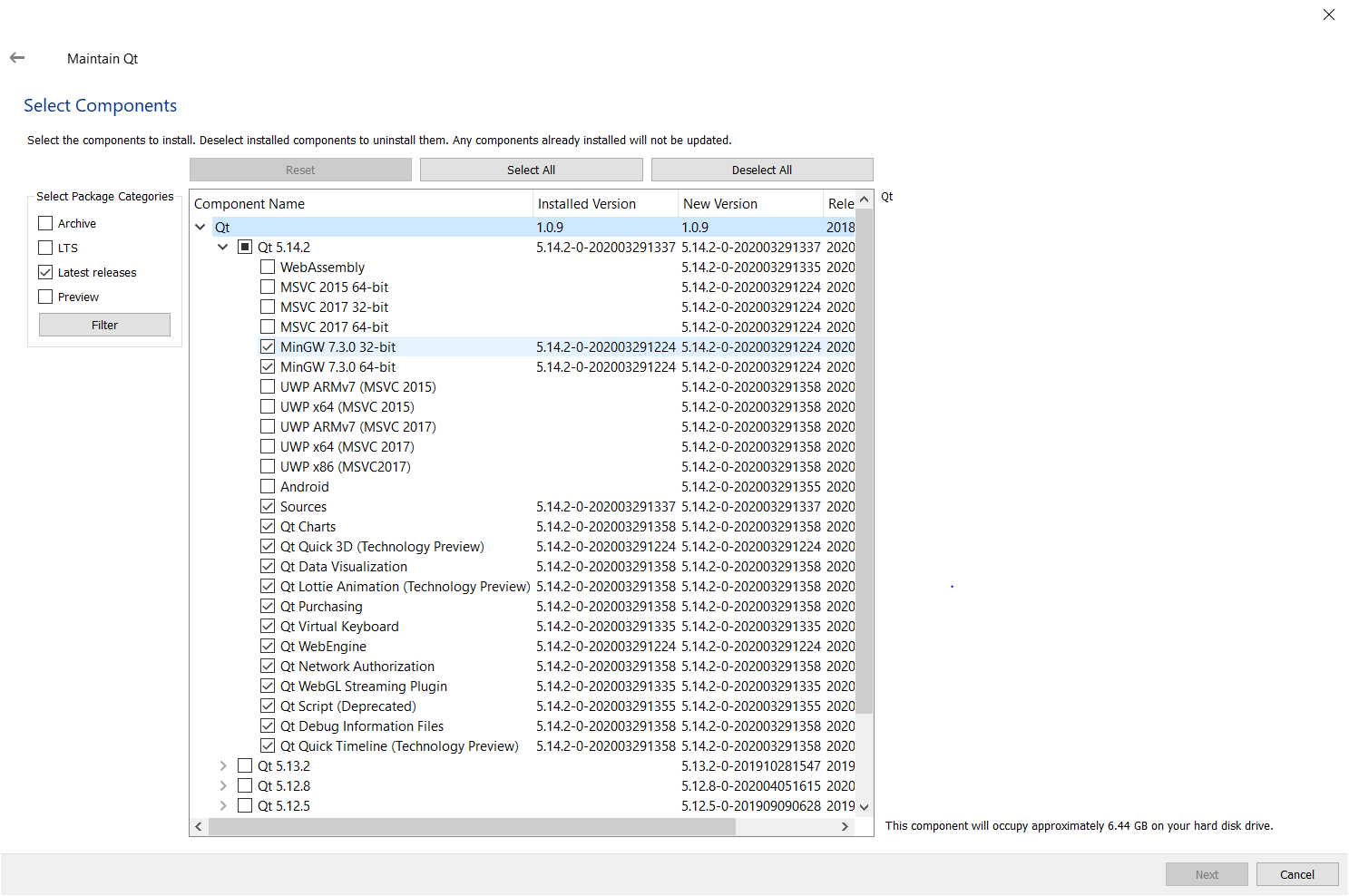
## Install Qt and Qt Creator

Download the Qt Online Installer from the Open Source Qt downloads website: <https://www.qt.io/download-open-source>

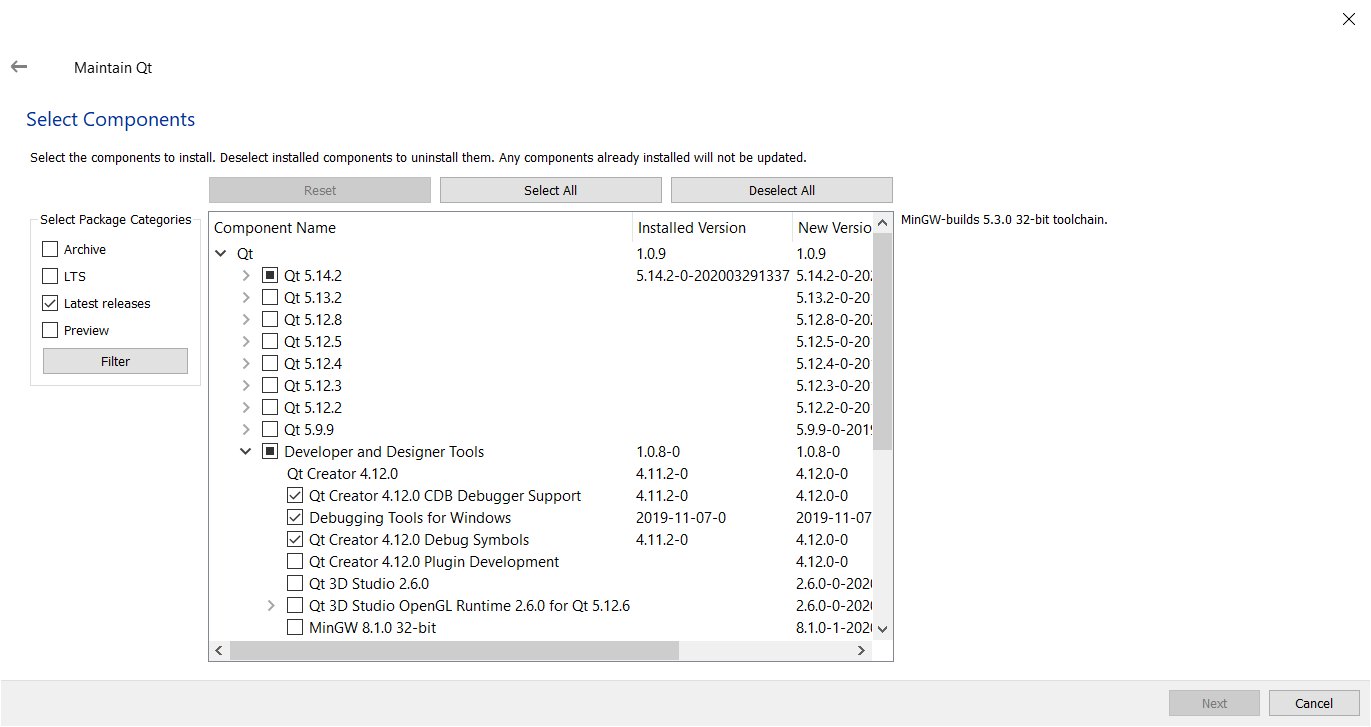
Launching the installer will bring you to a welcome page. On the following page, you will be asked to sign in with an existing Qt account or you will have the option to create a new account.

After signing in, a reminder of the open source usage obligations will be displayed. Select the check box to indicate you have read and agree with the obligations of using Open Source Qt.

Advance the process until you reach a screen which allows you to select components. Select Qt 5.14.2. This will auto select all the Qt 5.14.2 compilers and tools. However, you only need MinGW-32 compiler, MinGW-64 compiler, Sources, and all files that begin with Qt.



Also select Qt Creator 4.11.2, Qt Creator 4.11.2 CDB Debugger Support, and Debugging Tools for Windows from the Developer and Designer Tools.



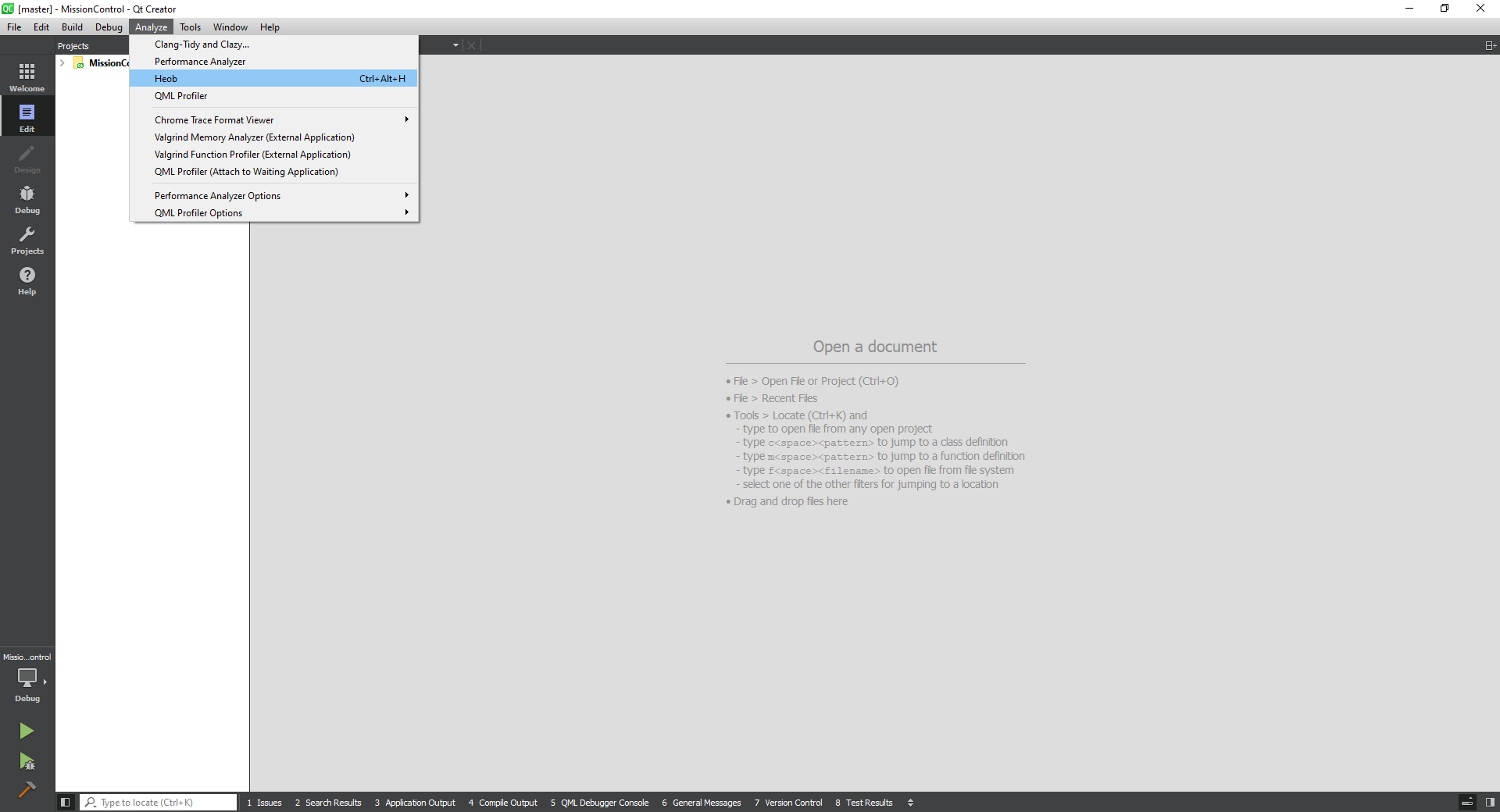
On the following page, read through and accept the terms and conditions of the LGPL, Python Software Foundation, and Microsoft Software license agreements. Advance the screens until you finish the installation process.

## Install Heob

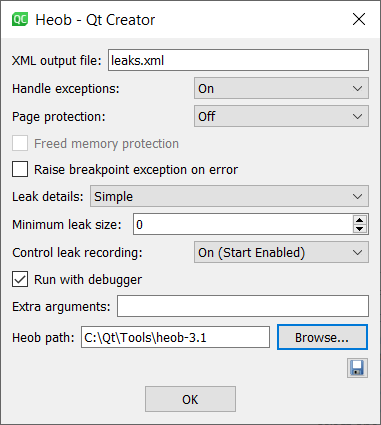
Download Heob from SourceForge (<https://sourceforge.net/projects/heob/>) or GitHub (<https://github.com/ssbssa/heob>). SourceForge provides an executable. If you download through GitHub, you will need to build Heob before it can be run from Qt Creator. Extract the zip file to the Qt\Tools directory. If you installed Qt using the Qt Online Installer, you will most likely find the directory through this path: C:\Qt\Tools.

To run Heob with MinGW, you will need to also provide Dwarfstack .dlls. Download them from here (<https://github.com/ssbssa/dwarfstack/releases>, unzip the files, and place them in the Qt/Tools/Heob folder.

Open Qt Creator. From the menu bar, select Analyze > Heob

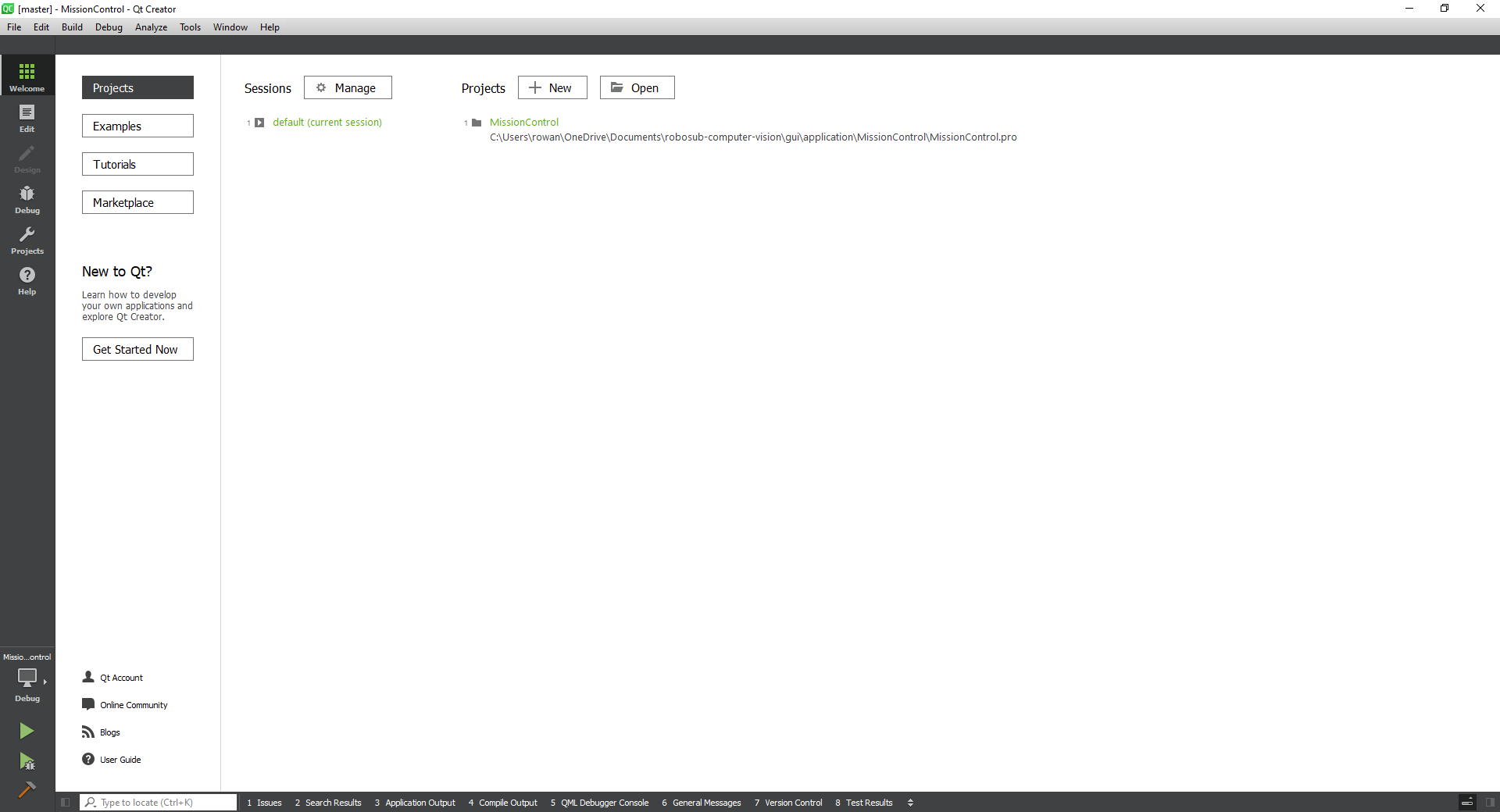


At the bottom of the Heob screen, select Browse and navigate to the C:\Qt\Tools directory. Select the Heob directory that contains the executables. Don’t forget to save the path by clicking the floppy disk icon below Browse.



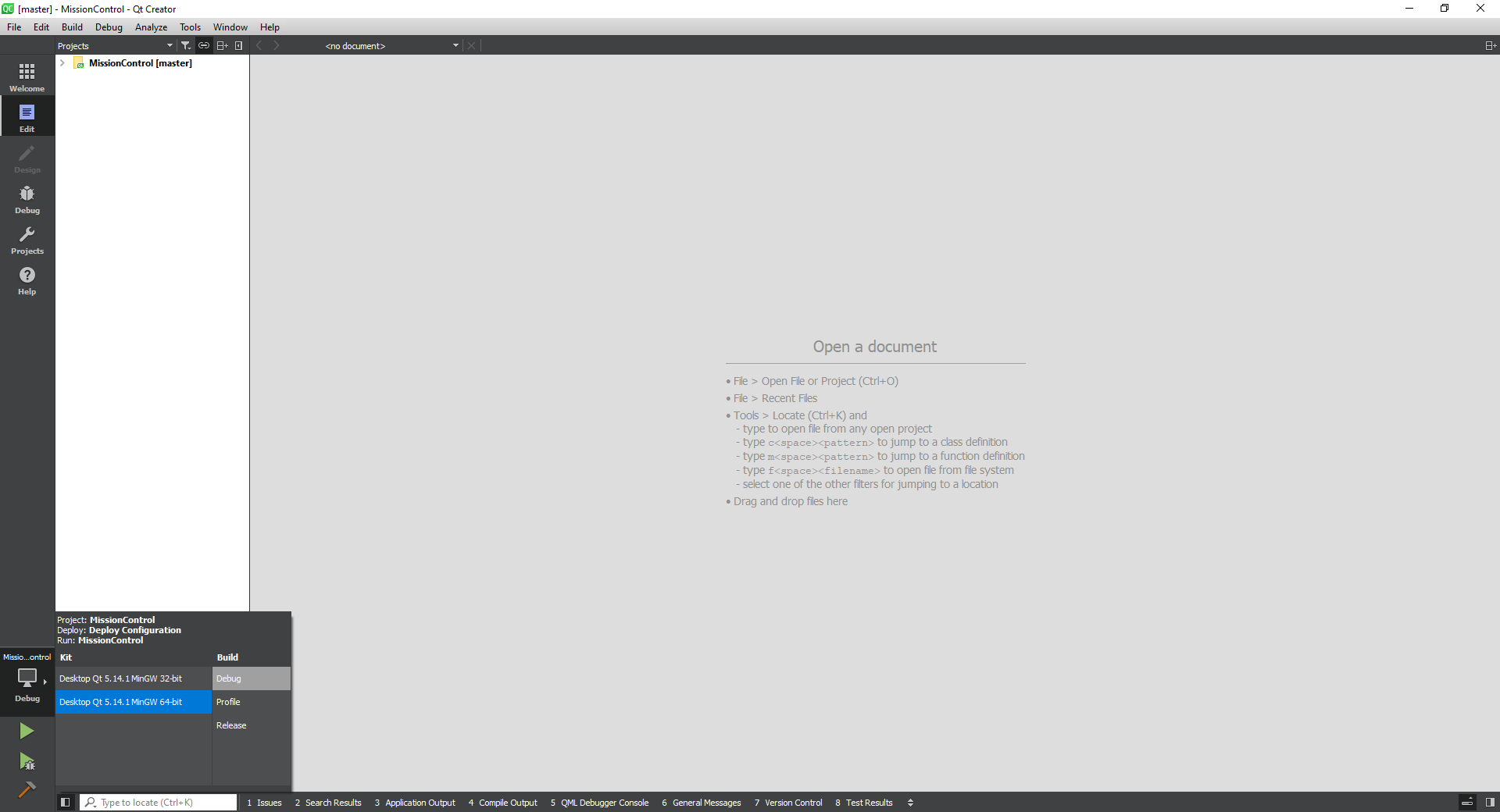
# Loading the Project File

Open Qt Creator. From the Welcome page, either select File > Open File or Project from the menu bar or Projects > Open from the widgets on the Welcome screen. Navigate to and select the MissionControl.pro file.



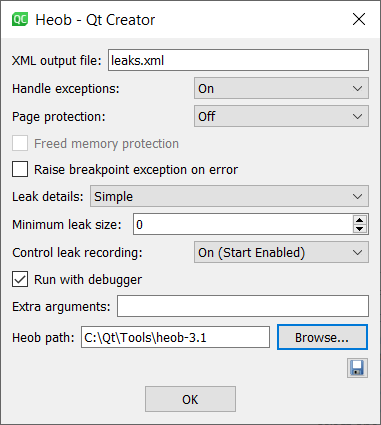
# Building and Running the Project

After opening the Mission Control project, locate the green arrow, green arrow with a bug, and hammer in the bottom left hand corner of the screen. The computer-like icon will designate which compiler kit your project will build with. Ensure that a MinGW kit is selected. The green arrow without a bug will run the application. The green arrow with a bug will run the application with debug. If the project has not been built before selecting one of the green arrows, Qt Creator will build the application before launching. The hammer icon will just build the application.



# Analyzing with Heob

To analyze the application with Heob, select Analyze > Heob from the menu bar in Qt Creator. Verify that the Heob path is correct. Select OK to launch the Heob tool.



Heob will run with the debugger and collect information about memory leaks. Similar to Valgrind, the application needs to successfully close for information to be collected. If you encounter a runtime error, you will need to re-launch Heob and the debugger.

# Target Color

The target color buttons feature a unique highlight when selected and emit their color when released. This signal is logged by the MainWindow class and pushed to the Mission export file upon mission confirmation.

# Target Shape

The target shape buttons emit their shape when released. This signal is logged by the MainWindow class and pushed to the Mission export file upon mission confirmation. Currently, only the Gate and Buoy buttons are supported. The other shape buttons are included by will require further development.

# Target Dimensions

The intent behind this section of application is to be able to select varying sized shapes. This was beyond the scope of the first version of the Mission Control application and never implemented.

# Task List

The task list is generated from a .txt file.

# Mission Export File

The mission file is comma delimited and in the following format:

<SHAPE>, <COLOR>

<TASK>, <TASK>, <TASK>, etc….